Rapport de projet « bataille navale »
Base de la programmation

Odic Oscar et Sinnadurai Thanan
S1D D11 2013

Table des matières

1 Notre démarche 2
1.1 Initialisation 2
1.2 Phase de tirs 4
2 Difficultés rencontrés et solutions adoptées 7
3 Manuel d'utilisation 10
3.1 Règles générales du jeu 10
3.2 Mode d'emploi du programme 11
3.2.1 Lancement du jeu 11
3.2.2 Significations des différentes cases des grilles 12
3.2.3 Utilisation du programme 13
4 Bilan de projet 15
5 Annexe : récapitulatif des fonctions/procédures 16
5.1 Fonctions principales (pour l'initialisation et la phase de tirs) 16
5.2 Fonctions pour l'initialisation (placement des vaisseaux) 20
5.3 Fonctions pour la phase de tirs 30

1 Notre démarche

Nous avons importé les bibliothèques suivantes :

`from upper import string` pour la fonction `upper('chaine')` qui retourne la chaîne avec les caractères en majuscule.
`from randint import random` pour la fonction `randint(nb1, nb2)` qui retourne un nombre compris entre nb1 et nb2 (inclus).
`from time import sleep` pour la fonction `sleep(temps)` qui gêle le terminal pendant le temps en paramètre.
`from os import system` pour la fonction `system('clear')` (ou `system('cls')` pour windows) afin de nettoyer le terminal.

Pour améliorer l'affichage nous avons décidé d'ajouter la ligne : `-*- coding: utf-8 -*-` pour pouvoir afficher des accents. On a aussi créer une fonction `separateur()` qui permet d'afficher un séparateur de plusieurs tirets.

Pour le changement de joueurs dans le cas de deux joueurs humain, une fonction `changement_joueur(nom_joueur)` a été créer permettant de nettoyer l'affichage du terminal et annoncer à quel joueur c'est le tour .

Nous avons divisé notre programme en deux grandes parties :

- La partie « initialisation », elle consiste à toute la partie avant la phase de jeu. C'est à dire le choix du nombre de joueurs, leurs noms, la difficulté si un seul joueur, et le placement des vaisseaux.

- La partie « phase de tirs », soit la phase ou les joueurs devrons couler les vaisseaux des autres.

1.1 Initialisation

Pour le début du programme nous avons décidé de créer une fonction `accueil_initialisation()` permettant d'accueillir le joueur pour la phase initialisation.
Cette fonction appel `system('clear')` (`system('cls')` pour Windows) pour nettoyer le terminal avant l'affichage du message de bienvenu.
La fonction appel ensuite « `regles_initialisation()` » qui rappel les règles pour la phase d'initialisation sous le choix du joueur.

Le programme appel ensuite les fonctions suivantes pour la préparation du jeu :

- `choix_nombre_joueurs()` : L'utilisateur choisit le nombre de joueurs, la valeur retourné est 1 ou 2.

- `choix_difficile()` Si le nombre de joueurs est de 1, l'utilisateur choisit la difficulté de l'ordinateur. Si la difficulté est de 0, la valeur retourné est « False » si elle est de 1, c'est « True ».

- `choix_noms(nombre_joueurs)` L'utilisateur choisit les noms des deux joueurs, sauf si le nombre_joueurs vaut 1, dans ce cas là, il ne choisit pas le nom du joueur 2 qui prend le nom de « ORDINATEUR », les noms sont retournés.

- `init_grille_vide()` Une grille de taille 10x10 est retourné avec les cases initialisé à '~~'.

Une grille d'exploration par joueur est ensuite initialisé avec cette fonction.

- `choix_automatique()` Pour chaque joueur, ce dernier choisit s'il veut placer automatiquement (renvoie « True ») ou manuellement (renvoie « False ») ses vaisseaux.

Le placement des vaisseaux peut alors commencer. Le programme appel une fonction `init_grille(auto, tableau_vaisseau)` pour chaque joueur qui initialise la grille personnelle (contient les vaisseaux), soit automatiquement (auto = True), soit manuellement. La fonction place les vaisseaux du « tableau_vaisseau » dans la grille.
Pour le changement de joueur (si l'on joue à deux) on utilise la fonction `changement_joueur()`
Si le joueur est l'ordinateur, les vaisseaux sont placés automatiquement.

Dans la fonction `init_grille(auto, tableau_vaisseau)`, on commence par appeler la fonction `init_grille()` pour initialiser la grille personnelle. La fonction appelle ensuite soit `placement_aleat_position_valide(grille, type_vaisseau)` (si placement automatique) soit `placement_manuel_position_valide(grille,type_vaisseau)` (si manuel) pour chaque vaisseau du tableau en paramètre dans init_grille. Ces deux fonctions retournent les coordonnés d'un vaisseau (ligne en lettre, colonne en chiffre et orientation en booléen, True pour vertical, False pour horizontal). Le premier les retournes aléatoirement, le deuxième demande les coordonnées aux joueur.

Ces deux fonctions appelles les fonctions `longueur_vaisseau(type_vaisseau)` permettant de retourner la longueur d'un vaisseau par rapport à son type, et `placement_valide(grille, lettre, chiffre, vertical, longueur)` qui retourne True si le placement du vaisseau est bien valide selon les règles, sinon False.

Le placement aléatoire utilise les fonctions :

- `choix_aleat_case(grille)` qui retourne une case aléatoire de la grille libre (case qui vaut « ~~ ») qui utilise `randint(0, 9)`.

- `entier_vers_bool(nombre)` qui retourne True pour 1 et
False pour 0 pour l'associer à la variable vertical à retourner.

Le placement manuel est une interface utilisateur utilisant :

- `affiche_grille(grille)` qui affiche la grille personnelle avant que le joueur choisisse les coordonnées du vaisseau.

- `choix_manuel_case()` et `choix_manuel_direction()` qui retournent les coordonnés d'une case et d'une orientation choisit par le joueur. Ces fonctions utilisent les fonctions `indice_colonne_valide(colonne)` et `indice_ligne_valide(ligne)` qui vérifient si les indices sont valides.

- `confirme_placement(lettre, chiffre, type_vaisseau, vertical)` qui utilise la fonction `confirme(val, message)` pour confirmer le choix du joueur.

Quand les coordonnées sont retournés par les fonctions de placement, on appel la fonction `ajoute_vaisseau(grille, lettre, chiffre, vertical, type_vaisseau)` qui ajoute le vaisseau dans la grille, cette fonction appelle `lettre_vers_indice(lettre)` pour convertir la lettre en indice pour la grille.

Les grilles personnelles sont initialisées, la partie Initialisation est donc terminée

1.2 Phase de tirs

Tout comme pour l'initialisation, nous avons crée une fonction `accueil_phase_de_tirs()` ayant pour but d'annoncer la phase de tirs et de rappeler, à l'aide de la fonction `regles_phase_de_tirs()`, les règles pour cette phase de jeu.

Vient alors le début du programme pour cette phase.

Nous avons initialisé deux tableaux `vaisseaux_j1`, `vaisseaux_j2` qui contiennent pour chaque case un nombre qui correspond à la quantité de vaisseaux restant pour un type donné.
Le tableau est de la forme :
`[nombre_sous-marin, nombre_frégates, nombre_destroyer, nombre_porte-avion]`
Il est donc initialisé avec les valeurs :
`[2, 2, 1, 1]`

Une autre variable `tour_joueur` permettant de savoir à quel joueur c'est le tour est initialisé à 0 (0 pour le joueur 1, 1 pour le joueur 2)

C'est alors que vient les tours de jeu.

Une boucle est donc mise en place. Avec comme condition d'entrée que les deux joueurs ont un nombre de vaisseaux différent de deux (aucun joueur n'a encore perdu).

D'où la création de la fonction `nombre_vaisseaux(vaisseaux)` qui calcule le nombre de vaisseaux restant à partir d'un tableau de nombre de vaisseaux d'un joueur et qui renvoi ce nombre.

Dans la boucle arrive deux cas.
Tout d'abord celui où c'est le tour du premier joueur, dans ce cas là, on lance la fonction `tour_de_jeu_humain()` contenant en paramètre le nom du joueur en cours, le nombre de joueurs, la grille d'exploration du joueur en cours, les grilles personnelles des deux joueurs, les coordonnées du tirs précédent ainsi que la taille du vaisseau coulé si l'adversaire en a tiré un au dernier tour (ces deux dernières variables sont initialisé à -1 pour le premier tour) ainsi que les deux tableaux pour les nombres de vaisseaux restants.

La fonction `tour_de_jeu_humain()` contient plusieurs fonctions pour le déroulement du tour.

La première est `messages_avant_tir()` qui affiche la situation, c'est à dire l'effet du tirs de l'adversaire (sauf au premier tour), ainsi que les vaisseaux restants pour les deux joueurs en fonction de leurs tirs.

Ensuite vient la fonction `affiche_grilles_joueur(grille_explo, grille_perso)` qui réutilise la fonction `affiche_grille()` pour afficher les deux grilles du joueur avant son tir. Elle sont aussi affiché après son tir, pour observer le résultat de son tour.

Pour entrer les coordonnées d'un tirs, nous avons crée la fonction `entrer_coordonnees()` qui, comme son nom l'indique, permet d'entrer les coordonnées d'un tir (deux caractères, une lettre puis un chiffre) et de les retourner. Dans cette fonction nous avons utilisé `verifie_coordonnees(coord)` qui verifie si les coordonnées entrées sont bien valide (retourne True pour valide, False sinon) c'est à dire présente dans la grille, et constitué de deux caractères. Pour vérifier sa présence dans la grille on a utiliser case_valide(lettre, chiffre) qui vérifie si la lettre et le chiffre sont valide pour entrer une coordonnée.

Le tir est ensuite effectué par la fonction `effectuer_tir(lettre, chiffre, grille_tir, grille_cible)` qui modifie la grille d'exploration du joueur et la grille personnelle de l'adversaire en fonction du tir : « TO » pour touché, « o » pour à l'eau.

La fonction `vaisseau_coule(lettre, chiffre, grille_tirs, grille_cible)` vérifie tout d'abord si le vaisseau est coule grace à quatre autres fonctions : `case_bateau_droite(lettre, chiffre, grille_cible)`, `case_bateau_gauche(lettre, chiffre, grille_cible)`, `case_bateau_bas(lettre, chiffre, grille_cible)`, `case_bateau_haut(lettre, chiffre, grille_cible)`, ces fonctions vérifies vers, le haut, le bas, la gauche, la droite si toutes les cases du vaisseau de cette direction sont touchés. Si le vaisseau est coulé, une fonction `remplace_coule(cases, grille_tirs, grille_cible)` remplace les cases touchés du vaisseau par « CO » dans les deux grilles.

Enfin, `message_apres_tir(coord_tir, grille_cible, taille_vaisseau)` affiche l'effet du tir du joueur en cours.

La fonction `tour_de_jeu_humain()` retourne ensuite les coordonnées du tir et la taille du vaisseau coulé s'il en a coulé un (sinon -1).

Le tour du joueur étant terminé, on lance la fonction `MaJ_vaisseaux(vaisseaux,taille_vaisseaux_coule)` qui met à jour le tableau du nombre des vaisseaux si un vaisseau est coulé, et la variable tour_joueur est modifié pour passer au tour de l'autre joueur.

Pour le tour du joueur 2, si le joueur est humain, le tour est exécuté comme pour le joueur 1. En revanche si le joueur est un ordinateur, la fonction `tour_de_jeu_humain()` est remplacé par `tour_de_jeu_ordinateur(grille_explo,grille_joueur_cible,difficile)`.

Si la difficulté est de 0, on utilise la fonction `coordonnees_aleat_vide(grille)` pour retourner une case aléatoire vide, cette fonction utilise `choix_aleat_case(grille)` pour chercher une case aléatoire, jusqu'à ce qu'il trouve une case qui vaut « ~~ » et la retourne.

Si la difficulté est de 1, on utilise la fonction `coordonnees_difficile(grille)`, qui utilise `cherche_coordonnees_touche(grille)` pour trouver une case qui vaut « TO », s'il ne la trouve pas, il retourne une case aléatoire à l'aide de `coordonnees_aleat_vide(grille)`
Si on la trouve, on utilise `coordonnees_proche(coord_initiale, grille_explo)` pour trouver une coordonnée proche qui pourrait contenir une case du vaisseau. On commence tout d'abord par chercher quelle est sa direction avec la fonction `cherche_direction(coord_initiale, grille)`, (0 pour vertical, 1 pour horizontal, -1 pour indéterminé.

Si la direction est verticale, on exécute la fonction `coordonnees_proche_verticale(coord, grille)` pour chercher une case proche verticalement, si horizontale, on cherche une case proche horizontalement avec la fonction `coordonnees_proche_horizontale(coord, grille)`. Si la direction est indéterminé, on essaye pour les deux directions.

Les coordonnées sont retournées, le tir est effectué avec `effectuer_tir(lettre, chiffre, grille_tir, grille_cible)`, puis on vérifie si le vaisseau est coulé avec `vaisseau_coule(lettre, chiffre, grille_tirs, grille_cible)`. On retourne ensuite du tour les coordonnées du tir pour le message du prochain tour ainsi que la taille du vaisseau si l'ordinateur en a coulé un, sinon -1.

Les nombres des vaisseaux peuvent être mise à jour avec `MaJ_vaisseaux(vaisseaux, taille_vaisseaux_coule)`.

Tant que les deux joueurs ont encore un nombre de vaisseaux différent de 0, on continue.

Sinon on sort de la boucle, et le dernier joueur à jouer gagne. On lance la fonction `fin_du_jeu(nom_gagnant)` qui annonce la fin du jeu et affiche le nom du gagnant.

2 Difficultés rencontrés et solutions adoptées

Durant ce projet nous avons eu plusieurs difficultés.

La première fut lorsque l'utilisateur entrait un caractères non numérique lorsqu'il devait entrer un entier.

En effet, par exemple, lorsqu'il est demandé à l'utilisateur d'entrer un entier compris entre 0 et 9 pour la colonne pour placer un vaisseau et que l'utilisateur entre une lettre, on obtient l'erreur :

```
ValueError: invalid literal for int() with base 10:
'A'
```

et le programme est stoppé.

Notre solution fut de concevoir une fonction qui convertit un caractère représentant un chiffre compris entre 0 et 9 en entier. Mais si ce caractère n'est pas un chiffre compris entre 0 et 9, la valeur retournée est -1.

Nous avons donc crée la fonction `car_vers_entier(car)` qui convertit le caractère en entier. Si ce caractère est un chiffre compris entre 0 et 9, l'entier retourné est celui retourné par la fonction `int`. Sinon, l'entier retourné est -1.

Donc si l'utilisateur entre une lettre, la fonction va lui assigner la valeur -1. Or l'utilisateur doit entrer une valeur comprise entre 0 et 9, donc le programme va afficher un message d'avertissement et redemander à l'utilisateur d'entrer un chiffre entre 0 et 9.

Nous avions aussi le problème de l'affichage sur le terminal pour le cas ou deux joueurs jouent. En effet, un joueur pouvait voir ce qu'il y avait affiché pendant le tour de l'autre joueur, et donc de voir, en l’occurrence, sa grille personnelle.

Après plusieurs recherches sur internet nous avons trouvé une fonction, disponible dans la librairie « os » qui permet d'exécuter la commande « clear » dans le terminal (ou « cls » pour Windows).

Il s'agit de la fonction `system()`.

Pour Linux, la commande est donc `system('clear')`.
Pour Windows , la commande est `system('cls')`.

Cette commande permet donc de nettoyer le terminal, et permet donc, à chaque changement de tour, d'empêcher le joueur de voir les données de son adversaire.

La plus grande difficulté de la partie initialisation était le placement des vaisseaux selon les règles du jeu. Il était assez compliqué de trouver un moyen d'empêcher le joueur de placer des vaisseaux qui se touchait ou se superposait ou sortait de la grille.

Nous avons commencer par nous occuper du cas ou le vaisseau sort de la grille. Dans la fonction `placement_valide(grille, lettre, chiffre, vertical, long)`, retournant un booléen indiquant s'il est possible ou non de placer un vaisseau, nous y avons indiqué :

```
if (((colonne + long) > 10) and not(vertical)) or (((ligne + long)
> 10) and vertical) :
     return False
```

Si l'orientation du vaisseau est horizontale, le numéro de colonne + la taille du vaisseau ne doit pas être plus grande que 10.
Si l'orientation du vaisseau est verticale, le numéro de ligne + la taille du vaisseau ne doit pas être plus grande que 10.

Pour vérifier si les vaisseaux se touchent ou se superposent, nous avons intégré dans `placement_valide(grille, lettre, chiffre, vertical, long)` la fonction `regles_case_valide(grille, lettre, chiffre)` qui vérifie que les règles de la case sont bien justes.

Elle est elle même décomposé en 2 fonction :

- `regles_colonne_valide(grille, lettre, chiffre)`

- `regles_ligne_valide(grille, lettre, chiffre)`

`regles_colonne_valide()` vérifie si la case en paramètre touche une case non vide dans l'orientation horizontale, si elle en touche une, on retourne False, sinon, on retourne True.

Pareil pour `regles_ligne_valide()` mais dans l'orientation verticale.

La fonction `placement_valide()` va donc essayer pour chaque case du vaisseau si elle touche d'autres cases non vide.
Si aucune d'elle ne touche d'autre case non vide, le placement est valide.

En effet, si un vaisseau est superposé sur un autre vaisseau, l'une des cases du nouveau vaisseau touche l'une de l'ancien vaisseau.

La première difficulté pour la phase de tirs était pour la vérification si un vaisseau était coulé ou non, et si oui, comment les modifier sur les grilles.

Notre solution fut de vérifier s'il existait un case non vide qui touchait une case « TO » sans elle même avoir été touché par le joueur, si oui, le vaisseau ne serait donc pas coulé.

Nous avons donc crée 4 fonctions dans la fonction `vaisseau_coule(lettre, chiffre, grille_tirs, grille_cible)` :

- `case_bateau_bas(lettre, chiffre, grille_cible)` : qui vérifie les cases allant vers le bas.

- `case_bateau_haut(lettre, chiffre, grille_cible)` : qui vérifie les cases allant vers le haut.

- `case_bateau_gauche(lettre, chiffre, grille_cible)` : qui vérifie les cases allant vers la gauche.

- `case_bateau_droite(lettre, chiffre, grille_cible)` : qui vérifie les cases allant vers la droite.

Pour la modification des cases, lorsque nous vérifions s'il existait une case non vide qui touchait le vaisseau, les fonctions `case_bateau_bas()`, `case_bateau_haut()`, `case_bateau_droite()`, `case_bateau_gauche()`, enregistrent les coordonnées des cases « TO » dans un tableau.

Si le tableau est coulé, toutes les cases « TO » contenues dans le tableau sont modifié en « CO » par la fonction `remplace_coule()`.

La deuxième difficulté de la phase était la difficulté 1 de l'ordinateur. Et plus particulièrement sa façon de tirer quand il avait touché un vaisseau.

Notre solution était de vérifier quel était sa direction en vérifiant si la deuxième case touchée (si il y en a une) se dirige horizontalement ou verticalement par rapport à la première avec la fonction `cherche_direction(coord_initiale, grille)`.

Ensuite, si l'orientation pouvait être su (2 cases touchés ou plus), nous utilisions soit `coordonnees_proche_horizontale(coord, grille)`, soit `coordonnees_proche_verticale(coord, grille)` en fonction de l'orientation.

Ces fonctions cherche respectivement à gauche ou à droite/en haut ou en bas une case vide « ~~ » proche d'une case « TO » et renvoie ses coordonnées.

Si l'orientation est indéterminée (1 case touché), les deux fonctions sont exécuté, et une des coordonnées trouvées est renvoyé.

Le dernier problème fut au niveau du placement de façon manuel, et plus particulièrement lorsqu'un joueur tirait sur une case déjà touché ou coulé. La case devenait « o ». donc lorsqu'une case « TO » était devenu « o », cela changeait la taille du vaisseau, et donc faussait le message sur les vaisseaux restants et dans le tableaux du nombres des vaisseaux.

Nous avons préféré retirer la possibilité d'entrer les coordonnées d'une case ou le joueur a déjà tiré. Sachant que ce genre de coup se faisait surtout par erreur ou lorsque les règles n'étaient pas comprises, il n'est pas utile de laisser le joueur pouvoir tirer au même endroit qu'il ai déjà tiré.

3 Manuel d'utilisation

3.1 Règles générales du jeu

Le jeu de la « bataille navale », appelé aussi « touché-coulé » est un jeu de société qui se joue à deux joueurs dans lequel les joueurs doivent placer leurs navires sur une grille « secrète » et doivent ensuite couler tous les navires adverses en touchant toutes les cases des navires à tour de rôle.

Le gagnant est le joueur ayant coulé tous les navires de son adversaire est déclaré vainqueur.

Chaque joueur possède une grille de 10 cases par 10 cases ou ces navires sont placés sans se superposer ou même se toucher :

- Deux sous-marins (2 cases)

- Deux frégate (3 cases)

- Un destroyer (4 cases)

- Un porte-avion (5 cases)

Pour la phase de tirs, chaque joueur doit l'un après l'autre doit choisir les coordonnées d'un tir constitué d'un indice de ligne et d'un indice de colonne.
Les effets du tir peuvent soit être « à l'eau » ou « touché ».

- À l'eau correspond à un tir n'ayant touché aucun navire

- Touché correspond à un tir ayant touché un navire

Lorsque toutes les cases d'un navire sont touchés, le navire est « coulé »
Le premier joueur ayant coulé tous les bateaux adverses est déclaré vainqueur.

3.2 Mode d'emploi du programme

3.2.1 Lancement du jeu

Sur Windows, téléchargez python pour Windows. Disponible sur http://www.python.org/download/.

Exécutez ensuite le fichier « bataille_navale_windows.py » à l'aide de python.exe (clic-droit/ouvrir avec/python.exe).

Sur Linux ouvrez le terminal et placez vous à l'aide de la commande « cd » dans le répertoire ou se trouve le fichier « bataille_navale_linux.py » et exécutez le programme à l'aide de la commande :

- « ./bataille_navale_linux.py »

Si le programme ne se lance toujours pas, vous pouvez aussi exécuter le programme à l'aide de la commande :

- « python bataille_navale_linux.py »

3.2.2 Significations des différentes cases des grilles

A chaque tour, le programme affichera la grille d'exploration et la grille personnelle du joueur. Par exemple :

```
Grille en jeu
  0  1  2  3  4  5  6  7  8  9
A ~~ ~~ ~~ ~~ ~~ ~~ ~~ FR TO FR
B ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
C ~~ ~~ FR ~~ ~~ o  ~~ ~~ ~~ ~~
D DT ~~ FR ~~ ~~ ~~ ~~ CO CO ~~
E DT ~~ FR ~~ o  ~~ ~~ ~~ ~~ ~~
F DT ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
G DT ~~ ~~ ~~ ~~ ~~ ~~ SM ~~ ~~
H ~~ ~~ o  ~~ ~~ ~~ ~~ SM ~~ ~~
I ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
J ~~ ~~ ~~ PA PA PA PA PA ~~ ~~


Grille de vos tirs
  0  1  2  3  4  5  6  7  8  9
A ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
B ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
C ~~ ~~ ~~ CO CO CO ~~ ~~ ~~ ~~
D ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
E ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
F ~~ ~~ o  ~~ ~~ ~~ ~~ o  ~~ ~~
G ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
H ~~ ~~ ~~ ~~ ~~ TO ~~ ~~ ~~ ~~
I ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
J ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
```

Les différentes types de cases sont :

- SM : sous-marin (navire de 2 cases)

- FR : frégate (navire de 3 cases)

- DT : destroyer (navire de 4 cases)

- PA : porte-avion (navire de 5 cases)

- o : tir à l'eau

- TO : case de navire touché

- CO : navire coulé

- ~~ : case vide pour la grille personnelle, case inconnue pour la grille d'exploration

3.2.3 Utilisation du programme

Au lancement du jeu, le programme vous donnera l'opportunité de lire les règles pour le placement de vaisseaux : Entrez « O » pour les lires, « N » sinon.

Le programme vous demandera ensuite un certain nombre de paramètres pour le fonctionnement du jeu.

Tout d'abord le nombre de joueurs. Si vous voulez jouer avec un autre joueur humain, entrez 2. Si vous entrez 1, vous allez jouer avec l'ordinateur.

Si vous avez choisit 1 au nombre de joueur, le jeu vous demandera la difficulté de l'ordinateur. Les niveaux de l'ordinateur sont les suivants :

- 0 : L'ordinateur tire dans une case aléatoirement, qu'il ai touché ou non un navire. En revanche, il ne tire jamais deux fois au même endroit.

- 1 : L'ordinateur tire aléatoirement dans une case ou il n'a jamais tiré. En revanche, quand il touche un navire, il concentre ses tirs suivants sur la zone ou est supposée se trouver le navire jusqu'à ce que le navire soit coulé.

Vous pouvez ensuite entrer vos noms...

Si le nombre de joueur est de 2, les deux noms serons demandés, s'il est de 1, il ne sera demandé qu'au joueur qui joue.

Chaque joueur devra ensuite choisir son mode de placement de navires, automatique place les navires automatiquement selon les règles, manuellement laisse le joueur choisir l'emplacement de ses navires.

Dans ce cas là, le jeu vous demandera la ligne de la case (lettre entre A et J), et la colonne de la case (entre 0 et 9), enfin il demandera la direction, horizontale (vers la droite), ou verticale (vers le bas).

Exemple :

```
Vous allez placer un vaisseau de type SM de longueur 2

  0  1  2  3  4  5  6  7  8  9
A ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
B ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
C ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
D ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
E ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
F ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
G ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
H ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
I ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~
J ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~ ~~


Entrez la ligne de la case (entre A et J inclus)
a
Entrez la colonne de la case (entre 0 et 9 inclus)
0
Entrez la direction du bateau (H pour horizontal, V pour vertical)
h
Valider le placement du vaisseau de type SM de longueur 2 [A0, horizontal] [O,N]?
o
```

Dans cet exemple, le navire se trouvera dans les cases A0 et A1.

On rappelle que vous devez placer 6 navires : deux sous-marins (2 cases), deux frégates (3 cases), un destroyer (4 cases) et un porte-avion (5 cases).

Les navires ne peuvent pas se chevaucher, ni se toucher et sont entourés par de l'eau (ils doivent êtres séparés par au moins une case représentant de l'eau).

Les placement effectué, la phase de jeu peut commencer...

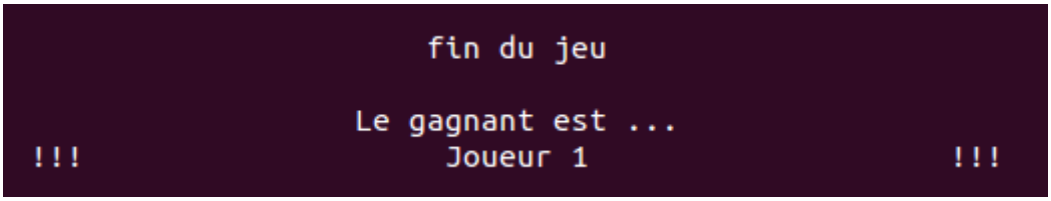
À chaque tour, chaque joueur devra entrer les coordonnées d'un tir.

Si le nombre de joueur est de 2, vérifiez bien à quel joueur c'est le tour (le jeu l'affiche). Quand c'est votre tour, le jeu vous demandera d'appuyer sur entrée pour jouer.

Prenez en compte des données affiché par le jeu avant d'entrer votre tir. Ces données prennent en compte l'effet du tour de l'adversaire avant vous ainsi que les navires restants pour les deux joueurs et vos grilles personnelles et d'exploration.

Il sera ensuite demandé d'entrer une coordonnée composée de une lettre et d'un chiffre. Ces coordonnées correspondent à un tir dans la grille de l'adversaire. La lettre (comprise entre A et J) correspond à la ligne, le chiffre correspond à la colonne. Vous ne pouvez pas tirer deux fois au même endroit. Les effets du tir vous serons ensuite affiché par texte et sur vos grilles.

Lorsque tous les navires d'un joueur serons coulé...



La fin du jeu est donc annoncé ainsi que le nom du gagnant.

4 Bilan de projet

Nous avons donc crée lors de ce projet un jeu de bataille navale sous le langage de programmation « python ».

Nous avons mis en œuvre l'ensemble de nos compétences apprises lors du semestre, tout en utilisant de nouvelles techniques après des recherches sur internet.

On note par exemple l'utilisation de « # -*- coding: utf-8 -*- » pour afficher des caractères utilisant des accents.

Ou l'utilisation de l'instruction « system('clear') » (« system('cls') » sur windows) pour nettoyer le terminal (apres avoir mentionner dans le programme « from os import system »).

Ce devoir nous à surtout permis de travailler sur un programme plus « sérieux » que ceux abordé en TD. En effet, c'était l'un de nos premier véritable projet en programmation. De plus cela nous a permis de travailler sur un programme véritablement utile (ici un jeu de bataille navale) contrairement aux exercices de TD.

Le projet nous à permis de mettre en place toutes nos compétences, et donc de les tester sur un programme plus important et de nous améliorer dans nos compétences en programmation.

5 Annexe : récapitulatif des fonctions/procédures

5.1 Fonctions principales (pour l'initialisation et la phase de tirs)

affiche_grille(grille)

Affiche une grille de taille 10x10 avec les colonnes numérotées de 0 a 9 et les lignes numérotées de A a J

@Input :
grille <list> un tableau a deux dimension de taille 10x10

@Output :
None

@Print :
Le contenu de la grille avec le numérotage des lignes et des colonnes

@Use :
None

car_vers_entier(car)

Convertit un caractère représentant un chiffre entre 0 et 9 en l'entier correspondant.
Si le caractère n'est pas un chiffre entre 1 et 9, la valeur retournée est -1, au lieu d'utiliser la fonction int(), arrêtant l'exécution du programme si l'utilisateur n'entre pas un nombre.

@Input :
car <str> caractère devant être convertit

@Output :
<int> entier entre 0 et 9 correspondant au caractère, sinon -1

@Print :
None

@Use :
None

case_valide(lettre, chiffre)

Renvoie True si la case fait partie de la grille, sinon False

@Input :
lettre <str> Lettre indiquant une ligne de la grille de jeu
chiffre <int> Chiffre indiquant une colonne de la grille de jeu

@Output :
<bool> True si la case est dans la grille, False sinon

@Print :
None

@Use :
indice_ligne_valide
indice_colonne_valide

changement_joueur(nomJoueur)

Efface le contenu affiche sur le terminal afin lors d'un changement de tour, pour empêcher le joueur de voir le contenu des grilles de l'autre joueur

@Input :
nomJoueur <str> Nom du Joueur dont c'est le tour

@Output :
None

@Print :
Efface le contenu affiche sur le terminal
Message de bienvenue au joueur dont c'est le tour

@Use :
system
sleep

choix_aleat_case(grille)

Renvoie une position aleatoire d'une case libre de la grille

@Input :
grille <list> grille de jeu

@Output :
<str> lettre entre A et J correspondant au numero de ligne
<int> chiffre entre 0 et 9 correspondant au numero de colonne

@Print :
None

@Use :
randint

indice_colonne_valide(chiffre)

Renvoie True si le chiffre correspond a un indice de colonne valide de la grille de jeu, et False sinon

@Input :
chiffre <int> indice de colonne

@Output :
<bool> True si l'indice est compris entre 0 et 9, sinon False

@Print :
None

@Use :
None

indice_ligne_valide(lettre)

Renvoie True si la lettre correspond a une indice de ligne valide de la grille de jeu, et False sinon

@Input :
lettre <str> indice de ligne

@Output :
<bool> True si l'indice est compris entre A et J, sinon False

@Print :
None

@Use :
lettre_vers_indice

lettre_vers_indice(lettre)

Renvoie l'indice de la ligne correspondant a la lettre

@Input :
lettre <str> la lettre correspondant a une ligne de la grille du jeu

@Output :
<int> indice de la ligne correspondante dans la grille du jeu, -1 si la lettre ne correspond a aucune ligne

@Print :
None

@Use :
upper

separateur()

Affiche un séparateur de 40 caractères "-"

@Input :
None

@Output :
None

@Print :
50 fois le caractere "-"

@Use :
None

5.2 Fonctions pour l'initialisation (placement des vaisseaux)

accueil_initialisation()

Affiche un message de bienvenue aux joueurs et affiche les règles du placement des vaisseaux s'ils le veulent.

@Input :
None

@Output :
None

@Print :
Efface le contenu affiche sur le terminal
Affiche le message de bienvenue et l'invite de l'affichage des règles du placement des vaisseaux

@Use :
regles_initialisation
separateur
system
sleep

ajoute_vaisseau(grille, lettre, chiffre, vertical, type_vaisseau)

Ajoute le vaisseau de type donnee, a la position donnee, dans l'orientation donnee, dans la grille

@Input :

grille	<list>	La grille du jeu
lettre	<str>	Lettre indiquant une ligne de la grille de jeu
chiffre	<int>	Chiffre indiquant une colonne de la grille de jeu
vertical	<bool>	True si l'orientation est verticale, False si elle est horizontale
type_vaisseau	<str>	type du vaisseau en deux lettres

@Output :
None

@Print :
None

@Use :
lettre_vers_indice
longueur_vaisseau

choix_automatique()

Interface utilisateur permettant de choisir s'il veut placer automatiquement ou pas les vaisseaux

@Input :
None

@Output :
<bool> True pour un placement automatique, False sinon

@Print :
messages d'invite et d'avertissement

@Use :
separateur
upper

choix_difficile()

Interface utilisateur permettant de choisir la difficulté de l'ordinateur

@Input :
None

@Output :
<bool> True si la difficulté est de 1, False si elle est de 0

@Print :
messages d'invite et d'avertissement

@Use :
separateur

choix_manuel_case()

Interface utilisateur permettant de choisir une case de la grille (pas de vérification d'occupation)

@Input :
None

@Output :
<str> lettre entre A et J correspondant au numéro de ligne
<int> chiffre entre 0 et 9 correspondant au numéro de colonne

@Print :
messages d'invite et d'avertissement

@Use :
car_vers_entier
indice_colonne_valide
indice_ligne_valide
upper

choix_manuel_direction()

Interface utilisateur permettant de choisir l'orientation du vaisseau

@Input :
None

@Output :
<bool> True si l'orientation est verticale, False si elle est horizontale

@Print :
messages d'invite et d'avertissement

@Use :
upper

choix_nombre_joueurs()

Interface utilisateur permettant de choisir le nombre de joueurs

@Input :
None

@Output :
<int> entier indiquant le nombre de joueur (1 ou 2)

@Print :
messages d'invite et d'avertissement

@Use :
car_vers_entier

choix_noms(nombre_joueurs)

Interface utilisateur permettant de choisir les noms des 2 joueurs pour le jeu

@Input :
nombre_joueur <int> Le nombre de joueur (s'il est de 1, le deuxième joueur a pour nom ORDINATEUR)

@Output :
<str> Le nom du premier joueur
<str> Le nom du deuxième joueur

@Print :
messages d'invite pour le choix des noms

@Use :
separateur

confirme(val, message)

Affiche le message de demande "message" pour la confirmation de la valeur "val". Renvoie True si celle est confirmée, sinon False

@Input :

val	<indeterminee>	La valeur a confirmer
message	<str>	Le message de demande

@Output :

<bool>	True si la valeur est confirmée, False sinon

@Print :

message de confirmation de la valeur et message d'avertissement

@Use :

upper

confirme_placement(lettre, chiffre, type_vaisseau, vertical)

Affiche une demande confirmation d'un placement d'un vaisseau de type "type_vaisseau", en ligne "lettre" et en colonne "chiffre" d'orientation "vertical"

@Input :

lettre	<str>	Lettre indiquant une ligne de la grille de jeu
chiffre	<int>	Chiffre indiquant une colonne de la grille de jeu
type_vaisseau	<str>	type du vaisseau a placer
vertical	<bool>	orientation du vaisseau, True si verticale, False sinon

@Output :

<bool>	True si le placement du vaisseau est confirme, False sinon

@Print :

Message de confirmation du placement

@Use :

confirme

entier_vers_bool(nb)

Convertit un entier en un booléen. 0 pour False, tous les autres entiers pour True.

@Input :
nb <int> Un entier correspondant a False s'il vaut 0, sinon True

@Output :
<bool> False si nb vaut 0, sinon True

@Print :
None

@Use :
None

init_grille(auto, vaisseaux)

Renvoie une grille initialisée avec les vaisseaux places

@Input :
auto <bool> placement aléatoire si True et placement manuel sinon
vaisseaux <list> liste des vaisseaux a positionner

@Output :
<list> matrice en deux dimensions correspondant a la grille initialisée

@Print :
message d'invite pour le placement manuel, affichage de la grille en construction

@Use :
ajoute_vaisseau
init_grille_vide
placement_aleat_position_valide
placement_manuel_position_valide
separateur

init_grille_vide()

Renvoie une grille vide de taille 10x10 remplis de "~~" pour chaque case (correspondant par défaut a une case vide)

@Input :
None

@Output :
<list> la grille de taille 10x10, avec chaque case initialise avec la valeur "~~"

@Print :
None

@Use :
None

longueur_vaisseau(type)

Renvoie la longueur d'un vaisseau de type donnée en paramètre

@Input :
type <str> type du vaisseau en 2 lettres ("SM" pour sous-marin, "FR" pour frégate, "DT" pour destroyer, "PA" pour porte-avion)

@Output :
<int> la longueur du vaisseau

@Print :
None

@Use :
None

placement_aleat_position_valide(grille, type_vaisseau)

Renvoie un placement valide pour le vaisseau choisi aléatoirement parmi les positions libres

@Input :

grille	<liste>	Une grille de taille 10x10
type_vaisseau	<str>	Le type du vaisseau en 2 lettres

@Output :

<str> La lettre correspondant a la ligne de la poupe du vaisseau

<int> Le chiffre correspondant a la colonne de la poupe du vaisseau

<bool> Le booléen vaut True si la direction de la proue est vers la verticale, et False sinon (vers la droite pour horizontale, vers le bas pour verticale)

@Print :

None

@Use :

choix_aleat_case
entier_vers_bool
placement_valide
randint

placement_manuel_position_valide(grille, type_vaisseau)

Renvoie un placement valide pour le vaisseau choisi manuellement par l'utilisateur et confirme par celui-ci

@Input :

grille	<list>	Une grille de taille 10x10
type_vaisseau	<str>	Le type du vaisseau en 2 lettres

@Output :

<str> La lettre correspondant a la ligne de la poupe du vaisseau

<int> Le chiffre correspondant a la colonne de la poupe du vaisseau

<bool> Le booleen vaut True si la direction de la proue est vers la verticale, et False sinon (vers la droite pour horizontale, vers le bas pour verticale)

@Print :

Messages d'invite et d'avertissement
Affichage de la grille avant choix
Messages de confirmation

@Use :

affiche_grille
choix_manuel_case
choix_manuel_direction
confirme
longueur_vaisseau
placement_valide
separateur

placement_valide(grille, lettre, chiffre, vertical, long)

Renvoie true si la position du vaisseau de taille long est bien valide selon les regles du jeu, sinon elle renvoie False

@Input :

grille	<list>	La grille de jeu de taille 10x10
lettre	<str>	Lettre indiquant la ligne de la grille de jeu
chiffre	<int>	Chiffre indiquant la colonne de la grille de jeu
vertical	<bool>	True si l'orientation est verticale, False si elle est horizontale
long	<int>	Longueur du vaisseau

@Output :

<bool>	Retourne True si la position du vaisseau est bien valide selon les regles du jeu, sinon False

@Print :

None

@Use :

lettre_vers_indice
regles_case_valide

regles_case_valide(grille, lettre, chiffre)

Renvoie True si la position de la case respecte bien les règles du jeu, False sinon

@Input :

grille	<list>	La grille de jeu de taille 10x10
lettre	<str>	Lettre indiquant la ligne de la grille de jeu
chiffre	<int>	Chiffre indiquant la colonne de la grille de jeu

@Output :

<bool>	True si position de la case respecte les règles du jeu, False sinon

@Print :

None

@Use :

regles_colonne_valide
regles_ligne_valide

regles_colonne_valide(grille, lettre, chiffre)

Renvoie True si la position de la case respecte les règles au niveau horizontal

@Input :
grille <list> La grille de jeu de taille 10x10
lettre <str> Lettre indiquant la ligne de la grille de jeu
chiffre <int> Chiffre indiquant la colonne de la grille de jeu

@Output :
<bool> True si position de la case respecte les règles du jeu au niveau horizontal, False sinon

@Print :
None

@Use :
indice_colonne_valide
lettre_vers_indice

regles_initialisation()

Affiche les règles ou pas pour le placement des vaisseaux selon l'envie des joueurs

@Input :
None

@Output :
None

@Print :
message d'invite pour le choix de voir ou non les règles
affichage des règles selon la réponse du message d'invite

@Use :
confirme
separateur

regles_ligne_valide(grille, lettre, chiffre)

Renvoie True si la position de la case respecte les règles au niveau vertical

@Input :
grille <list> La grille de jeu de taille 10x10
lettre <str> Lettre indiquant la ligne de la grille de jeu
chiffre <int> Chiffre indiquant la colonne de la grille de jeu

@Output :
<bool> True si position de la case respecte les règles du jeu au niveau vertical, False sinon

@Print :
None

@Use :
indice_ligne_valide
lettre_vers_indice

5.3 Fonctions pour la phase de tirs

accueil_phase_de_tirs()

Affiche un message d'accueil pour la partie "tour de jeu" aux joueurs et affiche les règles pour la phase des tirs.

@Input :
None

@Output :
None

@Print :
Efface le contenu affiche sur le terminal
Affiche le message d'accueil et l'invite de l'affichage des règles.

@Use :
regles_phase_de_tirs
separateur
system
sleep

affiche_grilles_joueur(grille_explo, grille_perso)

Affiche les deux grille(exploration et personnelle) du joueur

@Input :
grille_explo <list> grille d'exploration du joueur
grille_perso <list> grille personnelle du joueur

@Output :
None

@Print :
None

@Use :
affiche_grille
separateur

case_bateau_bas(lettre, chiffre, grille)

Conserve le nombre successif de case en dessous de la case de coordonnées lettre pour la ligne, chiffre pour la colonne, qui ne valent ni '~~', ni 'o ' dans une grille personnelle contenant des vaisseaux
Dans ces cases conservés :
Retourne le nombre de case de vaisseaux non touche
Retourne un tableau a deux dimensions contenant les coordonnées des cases de vaisseaux qui ont été touche

@Input :
lettre <str> lettre entre A et J correspondant a la ligne
chiffre <int> chiffre entre 0 et 9 correspondant a la colonne
grille <list> grille personnelle contenant des vaisseaux

@Output :
<int> nombre de case de vaisseaux non touche
<list> tableau a deux dimensions de coordonnées de case qui valent "TO"
Dans ce tableau :
- colonne d'indice 0 : indice de ligne des coordonnées
- colonne d'indice 1 : indice de colonne des coordonnées

@Print :
None

@Use :
case_vide
indice_ligne_valide
lettre_vers_indice

case_bateau_droite(lettre, chiffre, grille)

```
      Conserve le nombre successif de case a droite de la case de coordonnées
lettre pour la ligne, chiffre pour la colonne, qui ne valent ni '~~', ni 'o '
dans une grille personnelle contenant des vaisseaux
Dans ces cases conservés :
      Retourne le nombre de case de vaisseaux non touche
      Retourne un tableau a deux dimensions contenant les coordonnées des cases
de vaisseaux qui ont été touche

      @Input :
            lettre      <str>       lettre entre A et J correspondant a la ligne
            chiffre     <int>       chiffre entre 0 et 9 correspondant a la
colonne
            grille      <list>            grille personnelle contenant des
vaisseaux

      @Output :
            <int>       nombre de case de vaisseaux non touche
            <list>      tableau a deux dimensions de coordonnées de case qui
valent "TO"
                        Dans ce tableau :
                              - colonne d'indice 0 : indice de ligne des
coordonnées
                              - colonne d'indice 1 : indice de colonne des
coordonnées

      @Print :
            None

      @Use :
            case_vide
            indice_colonne_valide
            lettre_vers_indice
```

case_bateau_gauche(lettre, chiffre, grille)

```
      Conserve le nombre successif de case a gauche de la case de coordonnées
lettre pour la ligne, chiffre pour la colonne, qui ne valent ni '~~', ni 'o '
dans une grille personnelle contenant des vaisseaux
Dans ces cases conservés :
      Retourne le nombre de case de vaisseaux non touche
      Retourne un tableau a deux dimensions contenant les coordonnées des cases
de vaisseaux qui ont été touche

      @Input :
            lettre      <str>       lettre entre A et J correspondant a la ligne
            chiffre     <int>       chiffre entre 0 et 9 correspondant a la
colonne
            grille      <list>            grille personnelle contenant des
vaisseaux

      @Output :
            <int>       nombre de case de vaisseaux non touche
            <list>      tableau a deux dimensions de coordonnées de case qui
valent "TO"
                        Dans ce tableau :
                              - colonne d'indice 0 : indice de ligne des
coordonnées
                              - colonne d'indice 1 : indice de colonne des
coordonnées

      @Print :
            None

      @Use :
            case_vide
            indice_colonne_valide
            lettre_vers_indice
```

case_bateau_haut(lettre, chiffre, grille)

```
      Conserve le nombre successif de case au dessus de la case de coordonnees
lettre pour la ligne, chiffre pour la colonne, qui ne valent ni '~~', ni 'o '
dans une grille personnelle contenant des vaisseaux
Dans ces cases conservés :
      Retourne le nombre de case de vaisseaux non touche
      Retourne un tableau a deux dimensions contenant les coordonnées des cases
de vaisseaux qui ont été touche

      @Input :
            lettre      <str>       lettre entre A et J correspondant a la ligne
            chiffre     <int>       chiffre entre 0 et 9 correspondant a la
colonne
            grille      <list>            grille personnelle contenant des
vaisseaux

      @Output :
            <int>       nombre de case de vaisseaux non touche
            <list>      tableau a deux dimensions de coordonnées de case qui
valent "TO"
                        Dans ce tableau :
                              - colonne d'indice 0 : indice de ligne des
coordonnées
                              - colonne d'indice 1 : indice de colonne des
coordonnées

      @Print :
            None

      @Use :
            case_vide
            lettre_vers_indice
            indice_ligne_valide
```

case_vide(lettre, chiffre, grille)

Retourne True si la case de la grille correspondant a la ligne représente par la lettre et la colonne représente par le chiffre est vide ('~~' ou 'o ' ou 'TO' ou 'CO')

@Input :
lettre <str> lettre entre A et J correspondant a la ligne
chiffre <int> chiffre entre 0 et 9 correspondant a la colonne
grille <list> grille ou l'on vérifie la case

@Output :
<bool> True si la case vaut : - '~~'
- 'o '
- 'TO'
- 'CO'

@Print :
None

@Use :
lettre_vers_indice

cherche_coordonnees_touche(grille)

Renvoie les coordonnées d'une case ayant pour valeur 'TO' s'il y en a une, sinon, on renvoie -1

@Input :
grille <list> grille d'exploration de l'ordinateur

@Output :
<str> coordonnées en deux caractère, une lettre pour la ligne et un chiffre pour la colonne d'un case de valeur
'TO', -1 s'il en trouve pas

@Print :
None

@Use :
None

cherche_direction(coord_initiale, grille)

Renvoie la direction (0 pour verticale, 1 pour horizontale) d'un bateau en fonction des cases 'TO' s'il peut la trouver, sinon -1

@Input :
coord_initiale <str> coordonnées d'un case de valeur 'TO'
grille <list> grille d'exploration

@Output :
<int> direction du vaisseau : 0 pour verticale, 1 pour horizontale, -1 si impossible de savoir.

@Print :
None

@Use :
lettre_vers_indice

coordonnees_aleat_vide(grille)

Renvoie une coordonnée aléatoire valide pour une case vide dans une grille d'exploration pour l'ordinateur

@Input :
grille <list> grille d'exploration de l'ordinateur

@Output :
<str> coordonnées en deux caractères, une lettre pour la ligne et un chiffre pour la colonne

@Print :
None

@Use :
choix_aleat_case

coordonnees_difficile(grille_explo)

```
      Renvoie une coordonnée aléatoire valide pour le cas d'une difficulté de
1 :
      s'il y a une case 'TO', une coordonnée proche est renvoyé

      @Input :
            grille_explo      <list>      grille_d'exploration de l'ordinateur

      @Output :
            <str>             coordonnées en deux caractères, une lettre pour la
ligne et un chiffre pour la colonne

      @Print :
            None

      @Use :
            cherche_coordonnees_touche
            coordonnees_aleat_vide
            coordonnees_proche
```

coordonnees_proche(coord_initiale, grille)

```
      Renvoie les coordonnées valide d'une case vide proche d'une case de valeur
'TO'

      @Input :
            coord       <str>       coordonnées d'un case de valeur 'TO'
            grille      <list>      grille d'exploration

      @Output :
            <str>       coordonnées en deux caractères, une lettre pour la ligne
et un chiffre pour la colonne, ou l'int -1 si aucune case valide n'est trouve

      @Print :
            None

      @Use :
            cherche_direction
            coordonnees_proche_horizontale
            coordonnees_proche_verticale
            lettre_vers_indice
```

coordonnees_proche_horizontale(coord, grille)

Renvoie les coordonnées valide d'une case vide proche horizontalement d'une case de valeur 'TO'

@Input :
coord <str> coordonnées d'un case de valeur 'TO'
grille <list> grille d'exploration

@Output :
<str> coordonnées en deux caractères, une lettre pour la ligne et un chiffre pour la colonne, ou l'int -1 si aucune case valide n'est trouve

@Print :
None

@Use :
lettre_vers_indice

coordonnees_proche_verticale(coord, grille)

Renvoie les coordonnées valide d'une case vide proche verticalement d'une case de valeur 'TO'

@Input :
coord <str> coordonnées d'un case de valeur 'TO'
grille <list> grille d'exploration

@Output :
<str> coordonnées en deux caractères, une lettre pour la ligne et un chiffre pour la colonne, ou l'int -1 si aucune case valide n'est trouve

@Print :
None

@Use :
lettre_vers_indice

effectuer_tir(lettre, chiffre, grille_tir, grille_cible)

```
      Effectue le tir en fonction de la ligne(lettre) et de la colonne(chiffre)
en modifiant la case concerne dans la grille_tir et de la grille_cible en
fonction du vaisseau présent ou non sur cette case de la grille_cible :
      - 'o ' dans les deux grille si aucun vaisseau n'est présent dans
grille_cible
      - 'TO' dans les deux grills si un vaisseau est présent dans grille_cible

      @Input :
            lettre            <str>       lettre entre A et J correspondant a la
ligne
            chiffre           <int>       chiffre entre 0 et 9 correspondant a
la colonne
            grille_tir        <list>      grille d'exploration du joueur
effectuant le tir
            grille_cible      <list>      grille personnelle du joueur cible

      @Output :
            None

      @Print :
            None

      @Use :
            case_vide
            lettre_vers_indice
            upper
```

entrer_coordonnees()

```
      Interface utilisateur permettant d'entrer les coordonnees d'une case
(lettre puis chiffre)

      @Input :
            None

      @Output :
            <coord>     Coordonnées d'une case en deux caractères, le premier la
ligne sous forme de lettre, le deuxième la colonne sous forme de chiffre

      @Print :
            messages d'invite et d'avertissement

      @Use :
            verifie_coordonnees
            upper
```

fin_du_jeu(nom_gagnant)

```
      Affiche la fin du jeu et son gagnant

      @Input :
            nom_gagnant       <str>       Nom du gagnant ayant coulé tout les
vaisseaux adverses

      @Output :
            None

      @Print :
            Affiche la fin du jeu et le gagnant

      @Use :
            system
            sleep
```

MaJ_vaisseaux(vaisseaux, taille_vaisseaux_coule)

```
      Met a jour les informations du tableau "vaisseaux" si un vaisseau est
coule

      @Input :
            vaisseaux               <list>      tableau du nombre de vaisseaux
par type, SM, puis FR, puis DT et enfin PA
            taille_vaisseaux_coule  <int>       taille du vaisseau coule

      @Output :
            None

      @Print :
            None

      @Use :
            None
```

message_apres_tir(coord_tir, grille_cible, taille_vaisseau)

Affichage des effet du tir déroulé lors du tour

@Input :
coord_tir <str> coordonnées du tir du joueur
grille_cible <list> grille personnelle du joueur adverse
taille_vaisseau <int> taille du vaisseau si le vaisseau touche est coule

@Output :
None

@Print :
Affichage de l'effet du tir

@Use :
lettre_vers_indice
separateur
type_vaisseau
sleep

messages_avant_tir(coord_tir, taille_vaisseau, vaisseaux_joueur_tour, vaisseaux_joueur_cible, grille_perso)

Affichage de la situation : effet du tour de l'adversaire, nombre de vaisseaux restants pour vous et votre adversaire

@Input :
coord_tir <str> coordonnées du tir de l'adversaire
taille_vaisseau <int> taille du vaisseau coule si il y en a un qui l'a été ou tour précédent
vaisseaux_joueur_tour <list> nombre de vaisseaux du joueur pour chaque type [nb_SM, nb_FR, nb_DT, nb_PA]
vaisseaux_joueur_cible <list> nombre de vaisseaux du joueur adverse pour chaque type [nb_SM, nb_FR, nb_DT, nb_PA]
grille_perso <list> grille personnelle du joueur

@Output :
None

@Print :
Affichage des effets du dernier tours
Affichage du nombre de vaisseaux restants pour chaque joueur

@Use :
lettre_vers_indice
separateur
type_vaisseau
sleep

nombre_vaisseaux(vaisseaux)

Retourne le nombre total de vaisseaux restant en fonction du nombre de vaisseaux par type

@Input :
vaisseaux <list> nombre de vaisseaux du joueur pour chaque type [nb_SM, nb_FR, nb_DT, nb_PA]

@Ouptut :
<int> nombre total de vaisseaux restants

@Print :
None

@Use :
None

regles_phase_de_tirs()

Affiche les règles ou pas pour la phase de tirs selon l'envie des joueurs

@Input :
None

@Output :
None

@Print :
message d'invite pour le choix de voir ou non les règles
affichage des règles selon la réponse du message d'invite

@Use :
confirme
separateur

remplace_coule(cases, grille_tirs, grille_cible)

Remplace l’ensemble des cases indiques par le tableau a deux dimensions "cases" par la valeur 'CO' dans les grilles grille_tirs et grille_cible

@Input :

cases	<list>	Tableau a deux dimension contenant une coordonnée par ligne, sa ligne en colonne 0, et sa colonne en colonne 1
grille_tirs	<list>	Grille d'exploration du joueur dont c'est le tour
grille_cible	<list>	Grille personnelle du joueur cible

@Output :

None

@Print :

None

@Use :

None

tour_de_jeu_humain(nomJoueur, nb_joueurs, grille_explo, grille_perso, grille_joueur_cible, coord_tir_adverse, taille_si_coule, vaisseaux_joueur, vaisseaux_joueur_cible)

```
      Interface utilisateur pour un tour d'un joueur humain.
      Effectue un tour de jeu pour un joueur humain. L'affichage de la
situation, le choix des coordonnées du tirs, la vérification si le vaisseau est
coulé et l'affichage de l'effet du tir

      @Input :
            nomJoueur               <str>      Nom du joueur dont c'est le tour
            nb_joueurs              <int>      Nombre de joueurs humains qui
joue
            grille_explo            <list>     grille d'exploration du joueur
dont c'est le tour
            grille_perso            <list>     grille personnelle du joueur dont
c'est le tour
            grille_joueur_cible     <list>     grille personnelle du joueur
cible
            coord_tir_adverse       <str>      coordonnées du tir que
l'adversaire a fait au dernier tour
            taille_si_coule         <int>      taille du vaisseau coule par
l'adversaire au tour d'avant si il a coule un vaisseau, sinon -1
            vaisseaux_joueur        <list>     tableau d'entier, des effectifs
de chaque type de vaisseau du joueur dont c'est le tour
            vaisseaux_joueur_cible  <list>     tableau d'entier, des effectifs
de chaque type de vaisseau du joueur cible

      @Output :
            <str>                   coordonnées du tir effectue lors du tour
            <int>                   taille du vaisseau coule par le joueur lors
du tour s'il a coule un vaisseau

      @Print :
            affichage de la situation (tour de l'adversaire, nombre de vaisseaux
restants)
            message d'invite et d'avertissement
            affichage des effets du tir

      @Use :
            affiche_grilles_joueur
            changement_joueur
            effectuer_tir
            entrer_coordonnees
            message_apres_tir
            message_avant_tir
            separateur
            vaisseau_coule
```

tour_de_jeu_ordinateur(grille_explo, grille_joueur_cible, difficile)

Effectue un tour pour un ordinateur dans le cas ou un seul joueur joue

@Input :
grille_explo <list> grille d'exploration de l'ordinateur
grille_joueur_cible <list> grille personnelle du joueur cible
difficile <bool> difficulté (True pour 1, False pour 0)

@Output :
<str> coordonnées du tir effectue lors du tour
<int> taille du vaisseau coule par l'ordinateur lors du tour s'il a coule un vaisseau

@Print :
None

@Use :
coordonnees_aleat_vide
coordonnees_difficile
effectuer_tir
vaisseau_coule

type_vaisseau(taille_vaisseau)

Retourne le type du vaisseau correspondant a la taille passÉ en paramètre

@Input :
taille_vaisseau <int> taille du vaisseau

@Output :
<str> type de vaisseau

@Print :
None

@Use :
None

verifie_coordonnees(coordonnees)

Retourne True si les coordonnées sous la forme d'une chaîne a deux caractères (une lettre est un chiffre) représente bien une case valide

@Input :
coordonnees <str> chaîne de caractères

@Output :
<bool> True si le premier caractère est une lettre entre A et J et le second caractère est un chiffre entre 1 et 9

@Print :
None

@Use :
car_ver_entier
case_valide
upper

vaisseau_coule(lettre, chiffre, grille_tirs, grille_cible)

Retourne True si le vaisseau dont la case d'indices de ligne lettre, et de colonne chiffre inclue dans ce vaisseau est coule
Retourne False sinon, ou si la case n'a touche aucun vaisseau
Retourne aussi le nombre de cases du vaisseau si il est couche et remplace ses cases par 'CO' dans les deux grilles

@Input :
lettre <str> lettre entre A et J correspondant a la ligne
chiffre <int> chiffre entre 0 et 9 correspondant a la colonne
grille_tirs <list> grille d'exploration du joueur dont c'est le tour
grille_cible <list> grille personnelle du joueur cible

@Output :
<bool> True si le bateau est coule (toute ses cases sont touches), False sinon
<int> le nombre de cases du vaisseau s'il est touche, sinon -1

@Print :
None

@Use :
lettre_vers_indice
remplace_coule